Theoretical Computer Science 123 (1994) 415–426
Elsevier

Note

Restricted CRCW PRAMs

Naomi Nishimura*

Department of Computer Science, University of Waterloo, Waterloo, Ont., N2L 3G1, Canada

Communicated by M.S. Paterson
Received August 1991
Revised November 1992

Abstract

Nishimura, N., Restricted CRCW PRAMs, Theoretical Computer Science 123 (1994) 415–426.

We introduce a restriction on PRAMs which, when applied to the COMMON CRCW PRAM, yields a model that may be of use in determining the exact relationships between owner write, exclusive write, and concurrent write models. In partial work towards this goal, we present a lower bound of $\Omega(\log n)$ for computing OR on the restricted COMMON CRCW PRAM. In addition, we relate this model to the class of languages logspace-reducible to context-free languages by showing that any language accepted by such a PRAM in $O(\log n)$ time using a polynomial number of processors (with restricted instruction sets) is in this class. Finally, we demonstrate the sensitivity of the definition of the restriction to minor modifications.

1. Introduction

The study of various PRAM models [7–9] has resulted in an understanding of the impact of concurrent access to global memory on the speed of computations. Results concerning the complexities of various problems on various models have shown that the CREW PRAM [16] is strictly weaker than the CRCW PRAM [2]. The CREW PRAM and the CROW PRAM, defined by Dymond and Ruzzo [4], are equivalent when the number of processors need not be the same on each model [10]; when processor counts are bounded, their exact relationship is unknown.

Correspondence to: N. Nishimura, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, N2L 3G1. Email: nishi@maytag.waterloo.edu.

* Research supported by the Natural Sciences and Engineering Research Council of Canada.

SSDI 0304-3975(93)E0072-C

In this paper, we introduce *write-determination*, a new restriction on PRAMs. This notion arose in the study of asynchrony in shared memory models [11, 12]. In Section 2 we give a definition of this restriction, and show how both CROW PRAMs and PRIORITY CRCW PRAMs are not significantly weakened by its application. Then, in Section 3, we present a logarithmic lower bound on the computation of OR by a write-determined COMMON CRCW PRAM. Since the standard COMMON CRCW PRAM is capable of computing the OR of n inputs in constant time, the restricted model is closer in strength to a CREW PRAM. However, the exact relationship between the CREW PRAM and the write-determined COMMON CRCW PRAM is unknown when the number of processors is the same on both models.

The study of the hierarchy of PRAM models has been related to the study of certain classes of languages. In Section 4 we consider this relationship, and the relevance of write-determination to such considerations. In particular, we show that any language that is accepted by a write-determined COMMON CRCW PRAM using a polynomial number of processors with restricted instruction sets in logarithmic time is also logspace-reducible to the set of context free languages.

The definition of write-determination is in fact quite sensitive to minor modifications. We consider two such modifications in Section 5, and show how the resulting restrictions are much weaker than the original write-determination. Finally, in Section 6 we consider directions for further work.

2. Write-determination

We consider a PRAM restriction which concerns the state of a processor prior to its reading of a memory cell. Intuitively, an algorithm is write-determined if, before reading a memory cell, a processor knows when the cell was last written. In contrast, it is not difficult to alter a PRAM algorithm to ensure that a processor learns this information after reading a cell; a time-stamp can be inserted along with each write into a cell. More formally, write-determination is defined as follows.

Definition 2.1. A PRAM algorithm is said to be *write-determined* if for every processor P, every memory cell M, and every step T, if at time T processor P is in a state in which it is to read cell M, it can determine from its local information (before taking the step at time T) the largest $T' \leqslant T$ such that some processor wrote to M in step T'. The computation ends only when the result has been written into a specified answer cell or cells. Moreover, a processor that writes an answer cell must read it, so that all values used are read.

In the same way that we associate the EREW PRAM model with the set of legal algorithms on that model, we can define write-determined models to be models such that only write-determined algorithms are legal.

To illustrate the property of write-determination, we consider the constant-time algorithm for determining the OR of n bits on a CRCW PRAM. In the first step, each processor with a 1 writes a 1 into a collection cell. In the second step, all processors read the collection cell, writing a 0 in the answer cell if the collection cell is empty, and writing a 1 otherwise. This algorithm is not write-determined, because the processors reading the collection cell at time 2 do not know the largest $T' \leqslant 2$ such that some processor wrote to the collection cell in step T'. In fact, if a processor knew the time at which the collection cell was last written, it would then know the value of the computation without ever reading the cell.

Obliviousness is a special case of write-determination. In particular, any problem that can be solved on p-processor CRCW PRAM in time t by an oblivious algorithm can be solved on a p-processor write-determined CRCW PRAM with the same form of write-conflict resolution in time t. To see that this is true, we create a write-determined algorithm from the oblivious one. By definition, in the oblivious algorithm the pattern of accesses of the processors is fixed for all inputs. In particular, whenever a memory cell is read, the time at which the cell was last written is constant over all inputs. This information can be hard-wired into the processors, so that the write-determined restriction is satisfied.

Any CROW PRAM algorithm in which each processor has a single cell can be made into a write-determined algorithm by having each processor write to its cell at each step. By making further assumptions, it is possible to simulate an algorithm that runs in time t on a p-processor CROW PRAM in which there is a single cell per processor in $O(t+c)$ time on a p-processor CROW PRAM in which there are c cells per processor. Each processor in the simulating CROW PRAM first reads the values stored in its c cells and then writes the c values, encoded, into a single cell. In each of the subsequent steps, all c values are updated in a single write. A time complexity of t is possible if the values are not initially stored in global memory, since the initial reading stage can be avoided. The assumptions made for the simulation include increasing the number of bits held in a cell by a factor of c, increasing the number of bits written or read in one step by a factor of c, allowing constant time extraction of a particular value from the encoded group of c, and guaranteeing that each processor in the original algorithm knows the owner of each cell it reads.

Dymond and Ruzzo [4] show that a restricted CROW PRAM can simulate a generalized CROW PRAM without any increase in time, by a polynomial increase in the number of processors. The restrictions include the assignment of a single cell to a processor; the generalizations include the varying in time of the cells owned by a particular processor. The reader is referred to the original paper [4] for a full list of restrictions and generalizations.

Furthermore, by increasing the number of processors by the number of memory cells, it is possible to simulate a PRIORITY CRCW PRAM on a write-determined PRIORITY CRCW PRAM. We assign one caretaker-processor to each memory cell. At each step, each caretaker-processor reads the value contained in its cell, and then writes the same value back into the same cell. By making the caretaker-processors be

the processors of lowest priority, a caretaker-processor's write will succeed only if no other processor writes to the cell. The caretaker-processors guarantee that each cell is written at each step, and hence all algorithms are write-determined.

3. A lower bound on write-determined common

As a consequence of the following theorem, we prove a lower bound of $\Omega(\log n)$ on the time needed to compute the OR of n inputs on a write-determined COMMON CRCW PRAM, thereby separating the class of problems that can be solved in $o(\log n)$ on an n-processor write-determined COMMON CRCW PRAM from the class of problems that can be solved in $O(1)$ time on an n-processor COMMON CRCW PRAM. In the remainder of the paper, we use the notation WD-COMMON(t, p) to refer to the class of problems that can be solved in t time on a p-processor write-determined COMMON CRCW PRAM, with analogous notation for problems solved on other types of PRAMs. The proof of Theorem 3.1 makes use of techniques developed in the lower bound of Cook et al. [2] and extended by Nisan [10] and Dietzfelbinger et al. [3].

Theorem 3.1. *Any total Boolean function can be computed on a* CREW PRAM *in time* $O(t)$ *if and only if it can be computed on a write-determined* COMMON CRCW PRAM *in time* $O(t)$.

We first develop notation, and introduce the related measures of *decision tree complexity*, *certificate complexity*, and *block sensitivity*. It is by relating these notions with each other and with PRAM complexity measures that we are able to obtain our result.

A processor P can *distinguish* between two inputs at a particular time t if its state on one input is different from its state on the other input at that time. Similarly, we say that a memory cell M can distinguish X and X' at t if the state of M at t differs on X and X'.

For any input X, we can generate a distinct input $X^{(i)}$ by complementing the ith bit of X, so that $X^{(i)} = X_1 \ldots X_{i-1} \bar{X}_i X_{i+1} \ldots X_n$. More generally, for an input X and a set of bit positions, or *block*, A, $X^{(A)}$ is obtained from X by complementing the ith bit of X, for all $i \in A$. Processor P is said to be *sensitive to A on X at time t* if and only if P can distinguish X and $X^{(A)}$ at t. Again, we can make analogous definitions for the sensitivity of memory cell M.

We are now ready to define the various complexity measures. The *decision tree complexity* of a Boolean function f, or $D(f)$, is the minimal depth of a tree that computes f by checking a bit of f at each node, following a pointer to the child corresponding to the correct setting of the bit, and extracting the value of f from the leaf that is reached. A *certificate* is a block and a setting of bits in the block such that if any inputs X and X' agree with all bits in the block, then $f(X) = f(X')$. The *certificate*

complexity of f, $C(f)$, is simply the maximum over all inputs of the minimum size of a certificate. Finally, the *block sensitivity* of a Boolean function f, or $bs(f)$, is the maximum over all inputs X of the maximum number of disjoint blocks A such that for each A, $f(X)$ is distinct from $f(X^{(A)})$.

The known relations between the various complexity measures were established in the following set of theorems.

Theorem 3.2 (Nisan [10]). $bs(f) \leqslant C(f) \leqslant (bs(f))^2$.

Theorem 3.3 (Blum and Impagliazzo [1]). $C(f) \leqslant D(f) \leqslant (C(f))^2$.

By combining the results of Theorems 3.2 and 3.3, we can conclude that $\Theta(\log(bs(f))) = \Theta(\log(C(f))) = \Theta(\log(D(f)))$. The next two theorems relate PRAM classes to these measures.

Theorem 3.4 (Nisan [10]). *From any Boolean function f from $\{0,1\}^n$ to $\{0,1\}$, the time required to compute f on a CREW PRAM is in* $\Omega(\log(bs(f)))$.

Theorem 3.5 (Ragde [13]). *A Boolean decision tree of height h can be computed by a* CROW PRAM *in time* $O(\log h)$. *Any problem (partial or total) that can be solved by a* CROW PRAM *in time* $O(t)$ *can be solved by a decision tree of height* $O(2^t)$.

Since $\mathrm{CROW}(t,p) \subseteq \mathrm{CREW}(t,p)$, the time to compute a Boolean function f by a CROW PRAM or a CREW PRAM is in $\Theta(\log(bs(f))) = \Theta(\log(C(f))) = \Theta(\log(D(f)))$.

In Section 2, it was noted that a problem that can be solved on a p-processor CROW PRAM in time t using m memory cells can be solved in time t on a $(p+m)$-processor write-determined COMMON CRCW PRAM. Thus, to prove Theorem 3.1, it suffices to prove the following result, which holds regardless of the number of processors and memory cells involved.

Theorem 3.6. *For any total Boolean function f, the time required to compute f on a write-determined* COMMON CRCW PRAM *is in* $\Omega(\log(bs(f)))$.

Proof. Suppose that the block sensitivity of f is k, and on input X, f is sensitive to disjoint blocks $A_1, \ldots, A_k$. Following Nisan [10], we define a function g from $\{0,1\}^k$ to $\{0,1\}$ such that $g(z_1, \ldots, z_k) = f(w_1, \ldots, w_n)$, where $w_i = X_i^{(A_j)}$ if $i \in A_j$ and $z_i = 1$, and $w_i = X_i$ otherwise. The function g has sensitivity k, and can be computed by a COMMON CRCW PRAM at least as quickly as the function f. It will suffice to show a lower bound on the complexity of computing the function g.

Let $K(P,t,X)$ be the set of bits i such that processor P is sensitive to i at time t on input X, and let $L(M,t,X)$ be the set of bits i such that memory cell M is sensitive to i at time t on input X. Because of the write-determined restriction, whenever a memory cell is read, a processor knows at what time that cell was last written.

In order for an algorithm to compute correctly g, the processor determining the answer must be sensitive to at least k bits. Intuitively, the processor must "know" the values of all k bits; the proof proceeds by showing that a processor can at most double the number of bits it "knows" at each step. We can assume that a processor "remembers" previous states, so that if the state of processor P at time t is the same for X and Y, then the state of processor P at time $t-1$ is the same for X and Y. It is not difficult to see that the following lemma will suffice to complete the proof of the theorem.

Lemma 3.7. *On a write-determined* COMMON CRCW PRAM, *for each processor P, memory cell M, time t, and input X,*

$$|K(P,t,X)| \leqslant 2^t$$

and, if M is written at time t on input X, then

$$|L(M,t,X)| \leqslant 2^t.$$

Proof. The proof proceeds by induction on t. At the first step, each processor is sensitive to none of the bits, and each memory cell is sensitive to at most one of the bits. Suppose at time t on input X, processor P reads the value w from cell M. It is the state of processor P at $t-1$ on X that dictates which cell P reads, and from that state one can determine the time t' at which M was last written. Then w is the state of cell M at t' on X.

If $i \in K(P,t,X)$, we know that the state of P at t differs on X and $X^{(i)}$. If the state of P at $t-1$ differed on X and $X^{(i)}$, then $i \in K(P,t-1,X)$. If, on the other hand, the state of P at $t-1$ did not differ on X and $X^{(i)}$, then P read M at time t on both X and $X^{(i)}$. By write-determination, from the state of P at $t-1$, one can determine the time at which M was last written. Thus, since M was last written at time t' on input X, M was last written at time t' on input $X^{(i)}$. However, since the state of P at t differs on X and $X^{(i)}$, the value that P reads from M is different from w. Thus,

$$K(P,t,X) \subseteq K(P,t-1,X) \cup L(M,t',X)$$

and since $t' \leqslant t-1$ and M was written at time t' on input X, we have, as claimed,

$$|K(P,t,X)| \leqslant 2^{t-1} + 2^{t-1} = 2^t.$$

The argument is even simpler in the case in which at time t on input X, processor P does not read any value. It is not difficult to see that

$$|K(P,t,X)| \leqslant |K(P,t-1,X)| \leqslant 2^{t-1} < 2^t.$$

We now consider the sensitivity of a memory cell M that is written at time t on input X. Let $\mathscr{P}$ be the (nonempty) set of processors that write the value w into M at time t on input X. If $i \in L(M,t,X)$, then on the input $X^{(i)}$, the value w does not appear in cell M. Because the write-conflict resolution is COMMON, this means that none of the

processors in $\mathscr{P}$ write w into the cell M on $X^{(i)}$, and hence each of the processors in $\mathscr{P}$ is sensitive to i at t on X. More formally,

$$L(M,t,X)\subseteq \bigcap_{P\in\mathscr{P}} K(P,t,X).$$

By our proof for the reading case, we know that for each $P\in\mathscr{P}$, $|K(P,t,X)|\leqslant 2^t$, and hence the intersection of these sets is of size at most 2^t. This completes the proof of the lemma, and hence of the theorem. □

Fich and Ramachandran [6] have shown that the techniques can be generalized to handle functions from $\{0,1,\ldots,r\}^n$ to $\{0,1,\ldots,r\}$ in the same time bounds, where the number of processors used is a function of r.

Although Theorem 3.1 establishes the relationship between the complexity classes WD-COMMON(t,∞), CREW(t,∞), and CROW(t,∞), there remain questions to be answered concerning the relative strength of the classes when the number of processors is bounded. Just as the relation between CREW(t,p) and CROW(t,p) is unknown, the relationship of WD-COMMON(t,p) to these classes is also unknown.

4. Relations with LOGCFL

We can think of the PRAM computation of a Boolean function as the recognition of a language; we can thus relate PRAM classes to classes of languages. The classes LOGDCFL, LOGUCFL, and LOGCFL are the classes of languages that are log-space-reducible to, respectively, deterministic, unambiguous, and nondeterministic context-free languages. These were initially discussed by Sudborough [18], who showed that they are equivalent to the classes of languages recognizable by deterministic, unambiguous, and nondeterministic auxiliary push-down automata.

Whether or not LOGCFL equals LOGDCFL remains an open question. There has been some progress made concerning the relationships between languages and PRAM classes, as indicated in the following results. In some of the results, the instruction sets of the PRAM processors are restricted, e.g. to functions that are in AC^0 so that two n-bit numbers cannot be multiplied in constant time. For details on the restrictions, the reader is referred to the papers concerned.

Theorem 4.1 (Dymond and Ruzzo [4]). *A language L can be accepted by a* CROW PRAM *using a polynomial number of processors with restricted instruction sets in* $O(\log n)$ *time if and only if L is in* LOGDCFL.

Theorem 4.2 (Rytter [15]). *If a language L is in* LOGUCFL, *then L can be recognized by a* CREW PRAM *using a polynomial number of processors in* $O(\log n)$ *time.*

Ruzzo [14] relates alternating Turing machine complexity to LOGCFL and Stockmeyer and Vishkin [17] report a result of Ruzzo and Tompa relating alternating Turing machines to PRAMs. Combined, this work yields the following result.

Theorem 4.3. *If a language L is in* LOGCFL, *then L can be recognized by a* COMMON CRCW PRAM *using a polynomial number of processors with restricted instruction sets in* $O(\log n)$ *time.*

Although we can precisely define LOGDCFL in terms of the CROW PRAM, Theorems 4.2 and 4.3 each provide only half of a possible characterization of a language in terms of a PRAM model. By making use of the techniques of Theorem 4.1, we are able to simulate write-determined COMMON CRCW PRAMs directly by auxiliary nondeterminisitc PDAs.

Theorem 4.4. *If a language L is accepted by a write-determined* COMMON CRCW PRAM *using a polynomial number of processors with restricted instruction sets in* $O(\log n)$ *time, then L is in* LOGCFL.

Proof. The following is a sketch of the proof. For details, the reader is referred to the proof of Theorem 4.1 of [4], of which this is a modification. As in the original proof, the processors in the PRAM are spawned by a single original starting processor.

Given a language L that is accepted by a write-determined COMMON CRCW PRAM, it will suffice to show that the PRAM algorithm can be simulated in polynomial time on a logspace nondeterministic auxiliary PDA whose pushdown height is at most $O(\log^2 n)$. In essence, the PDA conducts a depth-first search of a DAG representation of the algorithm, starting with the output node.

The depth-first search consists of the implementation of three mutually recursive procedures, $state(t, P)$ returning the state of processor P at time t, $local(t, P, i)$ returning the contents of the ith local memory cell of processor P at time t, and $global(t, M)$ returning the contents of global memory cell M that was last written at time t. For example, to determine $local(t, P, i)$ it suffices to examine $state(t-1, P)$ and, if cell i was not updated at that step, to examine $local(t-1, P, i)$. The procedure $state(t, P)$ can also be implemented in a similar way, as discussed in [4].

The most difficult implementation is that of $global(t, M)$. To determine the contents of memory cell M, it is necessary to determine the state of the processor writing to M. In the deterministic simulation of the CROW PRAM by Dymond and Ruzzo [4], the processor writing to M is known to be the single processor that owns M. In our situation, several processors may have written M, but we do not know which ones. However, we can make use of nondeterminism and the write-determined property of the algorithm to solve the problem.

Our goal is to determine the value of M at time t. In our depth-first search of the DAG, we only attempt to determine the value contained in a memory cell at a particular time if there exists some processor that reads the value. In order to

determine the value written into M at time t', we use nondeterminism to guess a processor that writes to M at time t', and then, from the state of that processor, verify that the processor writes and extract the value written. Since the original algorithm performs on write-determined COMMON, all values written to M at time t' are the same, and hence it suffices to identify only one of the writing processors. □

It remains an open question whether or not the converse of Theorem 4.4, constituting a refinement of Theorem 4.3, is true. A theorem of this sort would relate the relationship between PRAM classes to the open question of the equivalence of LOGCFL and LOGDCFL.

5. Modifications on write-determination

The definition of write-determination is relatively sensitive. Seemingly minor modifications result in classes quite different from the write-determined classes. In this section, we consider two such modifications, and show in Theorems 5.3 and 5.4 that by slightly increasing the number of processors and memory cells, the resulting classes are as strong as the non-write-determined ones. In contrast, the lower bound of Theorem 3.6 that separates write-determined COMMON from COMMON holds regardless of the number of processors and memory cells involved.

A *semi-write-determined* algorithm differs from a write-determined one in that instead of knowing exactly when a cell was last written, a processor knows a pair of times, one of which is the time at which the cell was last written. In a *conditional-write-determined* algorithm, a processor knows when a cell was last written only if the cell has been written at least once during the computation. More formally, we have the following definition.

Definition 5.1. A PRAM algorithm is *semi-write-determined* if for every processor P, every memory cell M, and every step T, if at time T processor P is in a state in which it is to read cell M, it can determine from its state two times $T' \leqslant T$ and $T'' \leqslant T$ such that either T' or T'' is the latest time at which some processor wrote to M.

Definition 5.2. A PRAM algorithm is *conditional-write-determined* if for every processor P, every memory cell M, and every step T, if cell M has ever been written, and if at time T processor P is in a state in which it is to read cell M, then processor P can determine from its state at time T the largest $T' \leqslant T$ such that some processor wrote to M at step T'.

In the following, we refer to the *access type* of a PRAM when we specify the degree of concurrency allowed in reads and writes and the write-conflict resolution method, if write conflicts are allowed. Thus, a COMMON CRCW PRAM and a write-determined COMMON CRCW PRAM are PRAMs of the same access type, whereas a CROW PRAM and a CREW PRAM are of different access types.

Theorem 5.3. *Any problem that can be solved in time t on a* PRAM *of a particular access type using p processors and m memory cells can be solved in time 2t on a semi-write-determined* PRAM *of the same access type using p+m processors and m memory cells.*

Proof. Of the $p+m$ processors of the semi-write-determined PRAM, p of the processors are *simulating-processors*, each assigned to simulate a different processor in the original machine, and m of the processors are *caretaker-processors*, each assigned to a different memory cell. Step i of the original machine is divided into steps $2i-1$ and $2i$, where the odd steps are taken by the p simulating-processors and the even steps are taken by the m caretaker-processors.

More specifically, a read of cell M by processor P at time i in the original algorithm is simulated by a read of the cell M at time $2i-1$. A write of value v into cell M by processor P at time i in the original algorithm is simulated by a write of v into cell M at time $2i-1$. At each even timestep, each caretaker-processor will read the cell to which it is assigned, and rewrite the value that it reads. Consequently, before a simulating-processor reads a cell, it knows that the cell was last written at the previous timestep; before a caretaker-processor reads a cell, it knows that the cell was last written during one of the previous two timesteps. Thus, the semi-write-determined condition is satisfied. Since values held in cells at time $2i$ in the new algorithm are identical to the values held at time i in the original algorithm, it is not difficult to see that the original algorithm is correctly simulated. This completes the proof of the theorem. □

Theorem 5.4. *Any problem that can be solved in time t on a* PRAM *of a particular access type using p processors and m memory cells can be solved in time 2t on a conditional-write-determined* PRAM *of the same access type using p+m processors and mt memory cells.*

Proof. As in the proof of Theorem 5.3, p of the processors are simulating-processors, m of the processors are caretaker-processors, and each step of the original machine is simulated in two steps of the new machine. In addition, there are t copies of the original memory, denoted as $M^{(1)}, \ldots, M^{(t)}$.

As before, reads and writes at time i in the original algorithm are simulated at time $2i-1$ in the new algorithm. At time $2i-1$, a simulating-processor will read from a cell in $M^{(i)}$ and write into a cell in $M^{(i+1)}$. The caretaker-processors are responsible for making sure that the contents of the memory cells are not lost from step to step, by storing the values in local memory. Specifically, a caretaker processor will read a cell in $M^{(i+1)}$ and then write into that cell either the value just read, or, if no value had been written there, the corresponding value from $M^{(i)}$ (which was stored in local memory). Since a write occurs at step $2i$ in either case, the condition-write-determined condition is satisfied for the simulating-processors. The condition is also satisfied by each read of a caretaker-processor, since a cell in $M^{(i+1)}$ was either written during step $2i-1$ or not at all. □

6. Conclusions and open questions

We have shown that the class WD-COMMON($o(\log n), n$) is strictly weaker than the class COMMON($O(1), n$). Write-determination is inherent in a CROW PRAM due to the weakness of the model, and in a PRIORITY CRCW PRAM due to its strength. For other classes of PRAMS, the relationships between write-determined and non-write-determined PRAM classes are unknown.

Although results obtained for partial domain problems are not as strong as those obtained for full domain problems, they may still provide an indication of the strengths of various classes. The problem XREP is defined in [5] as follows: given an assignment of bits to processors such that at least one processor gets a 1, determine a single processor that has a 1. This problem can be solved in $O(1)$ time by a write-determined ARBITRARY CRCW PRAM. In the proof of the lower bound on the COMMON CRCW PRAM given for the full domain version of XREP, the all zero input is removed. Consequently, the same proof holds for the partial domain problem, separating the two models in this domain.

If the numbers of processors in the various models may differ, the CROW PRAM is as strong as the write-determined COMMON CRCW PRAM. For equal numbers of processors, we consider partial domain problems. Snir [16] considers the problem of determining RANK, the rank of an element in a sorted list. This problem can be solved in $O(1)$ time on a write-determined COMMON CRCW PRAM, a write-determined CREW PRAM, or a CREW PRAM. However, for a CROW PRAM, the complexity of RANK is $\Theta(\log \log n)$. The upper bound is obtained by representing the problem as a decision tree of height $\log n$; the lower bound arises from a simple information theory lower bound and its impact on decision tree height. It is possible to determine in constant time on a CREW PRAM the OR of n inputs where at most one input has the value 1. However, by an argument similar to that used in Theorem 3.6, this problem requires $\Omega(\log n)$ time on a write-determined COMMON CRCW PRAM. The incomparability of the CREW PRAM and the write-determined COMMON CRCW PRAM has yet to be shown.

References

[1] M. Blum and R. Impagliazzo, Generic oracles and oracle classes, in: *Proc. 28th Annu. IEEE Symp. on the Foundations of Computer Science* (1987) 118–126.

[2] S. Cook, C. Dwork and R. Reischuk, Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.* **15** (1986) 87–97.

[3] M. Dietzfelbinger, M. Kutyłowski and R. Reischuk, Exact time bounds for computing Boolean functions on PRAMs without simultaneous writes, in: *Proc. 2nd Annu. ACM Symp. on Parallel Algorithms and Architectures* (1990) 125–135.

[4] P. Dymond and W.L. Ruzzo, Parallel RAMs with owned global memory and deterministic context-free language recognition, in: *Proc. 13th Internat. Colloq. on Automata, Languages, and Programming* (1986) 95–104.

[5] F. Fich, P. Ragde and A. Wigderson, Relations between concurrent write models of parallel computation, *SIAM J. Comput.* **17** (1988) 606–627.
[6] F. Fich and V. Ramachandran, Lower bounds for parallel computation on linked structures, in: *Proc. 2nd Annu. ACM Symp. on Parallel Algorithms and Architectures* (1990) 109–116.
[7] S. Fortune and J. Wyllie, Parallelism in random access machines, in: *Proc. 10th Annu. ACM Symp. on the Theory of Computing* (1978) 114–118.
[8] L. Goldschlager, A unified approach to models of synchronous parallel machines, *J. ACM* **29** (1982) 1073–1086.
[9] G. Lev, N. Pippenger and L. Valiant, A fast parallel algorithm for routing in permutation networks, *IEEE Trans. Comput.* **C-30** (1981) 93–100.
[10] N. Nisan, CREW PRAMs and decision trees, in: *Proc. 21st Annu. ACM Symp. on the Theory of Computing* (1989) 327–335.
[11] N. Nishimura, Asynchronous shared memory parallel computation, in: *Proc. 2nd Annu. ACM Symp. on Parallel Algorithms and Architectures* (1990) 76–84.
[12] N. Nishimura, Asynchrony in shared memory parallel computation, Ph.D. Thesis, University of Toronto, 1991.
[13] P. Ragde, personal communication, 1989.
[14] W.L. Ruzzo, Tree-size bounded alternation, *J. Comput. System Sci.* **21** (1980) 218–235.
[15] W. Rytter, Parallel time $O(\log n)$ recognition of unambiguous context-free languages, *Inform. and Comput.* **73** (1987) 75–86.
[16] M. Snir, On parallel searching, *SIAM J. Comput.* **14** (1985) 688–708.
[17] L.J. Stockmeyer and U. Vishkin, Simulation of parallel random access machines by circuits, *SIAM J. Comput.* **13** (1984) 409–422.
[18] I.H. Sudborough, On the tape complexity of deterministic context-free languages, *J. Assoc. Comput. Mach.* **25** (1978) 405–424.